Feeling Comfortable with Logic Analyzers

HEWLETT PACKARD

About This Book ...

You need to look simultaneously at the inputs and outputs of a 16-bit counter to determine a timing error and you have only a 2-channel scope. How do you look at them all? You've just developed timing diagrams for a board full of digital circuitry. How do you verify them? An intermittent glitch appears to occur on one of the data lines of your microprocessor system, causing the processor to get incorrect data. You can't trigger on just the glitch with your analog oscilloscope. What do you use to capture and analyze it?

With the wrong tool, solving these kinds of problems can be very time consuming. The more time-consuming the problem, the more nervous the boss becomes. In such cases, being a hero may depend on knowing which tool can get the job done quickly. For the above problems, the best solution is the logic analyzer.

This little book is intended as a quick overview of logic analyzer basics. With it, we want to cut down on the time investment needed to learn a new instrument. It doesn't cover many detailed measurements, but it does give you a good idea of what a logic analyzer can do. Since we are all pressed for time in our jobs, we wanted to provide you with a short simple book that doesn't take a lot of time to read but still covers the basics. We talk about questions like "Why should I bother with a logic analyzer?" and "What will one do for me?". This is not a manual for any specific analyzer, even though the examples are taken from Hewlett-Packard products. It will help you to understand the contributions of logic analyzers.

Feeling Comfortable With Logic Analyzers

HEWLETT PACKARD

Printing History Second Edition

April 1988 Printed in U.S.A.

Book Part Number 5954-2686

Contents

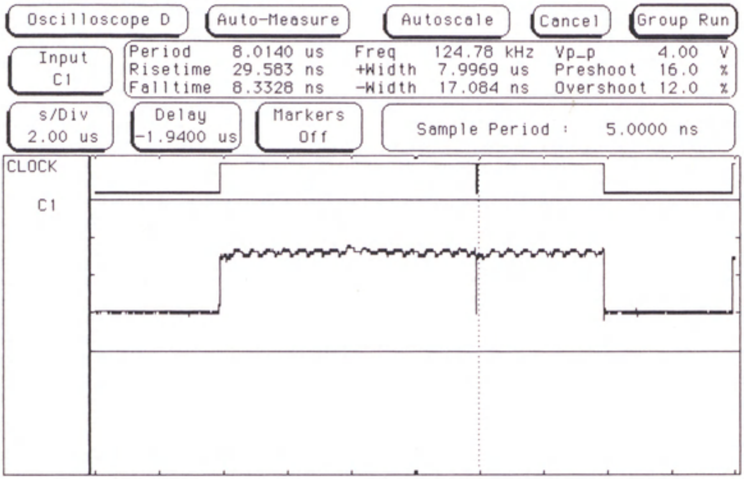
1 Oscilloscope or Logic Analyzer?

When given the choice between using an oscilloscope or a logic analyzer, many people will choose an oscilloscope. Why? Because a scope is more familiar to most users. There is one on virtually every engineer's bench, and it is relatively easy to use. It is one of the most "general purpose" of all electronic instruments. However, it has some shortcomings that limit its usefulness in some applications. A logic analyzer may yield more useful information in many of these applications. However, since a logic analyzer is "tuned" to the digital world, it doesn't have as broad a use as an oscilloscope. Because of some overlapping capabilities between a scope and a logic analyzer, either may be used in some cases. How do you determine which is better for your application? The next few paragraphs give some basic guidelines.

When Should I Use a Scope?

- When you need to see small voltage excursions on your signals.
- When you need high time-interval accuracy.

Generally, an oscilloscope is the instrument to use when you need high vertical or voltage resolution. To say it another way, if you need to see every little voltage excursion, like those shown below on the bottom waveform, you need a scope. Many scopes, including the new generation digitizing ones, can also provide very high time-interval resolution. That is, they can measure the time interval between two events with very high accuracy. Overall, an oscilloscope is to be used when you need parametric information.

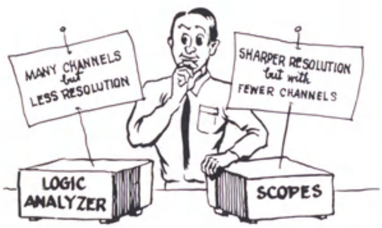


When Should I Use a Logic Analyzer?

- When you need to see lots of signals at once.
- When you need to look at signals in your system the same way your hardware does.
- When you need to trigger on a pattern of highs and lows on several lines and see the result.

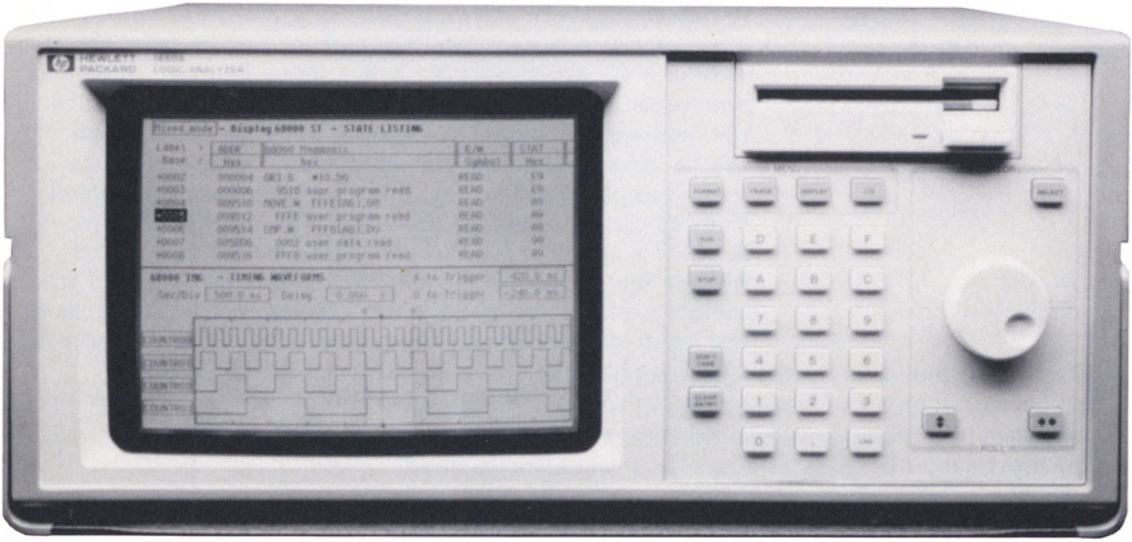
Logic analyzers grew out of oscilloscopes. They present data in the same general way that a scope does; the horizontal axis is time, the vertical axis is voltage amplitude. But a logic analyzer does not provide as much voltage resolution or time interval accuracy as its cousin, the oscilloscope. It can capture and display eight or more signals at once, something that scopes cannot do. A logic analyzer reacts the same way as your logic circuit does when a single threshold is crossed by a signal in your system. It will recognize the signal to be either low or high. It can also trigger on patterns of highs and lows on these signals. So when do you use a logic analyzer? When you need to look at more lines than your oscilloscope can show you, provided you can live without ultra-precise time interval information. If you need to look at every little transition on the waveform, a logic analyzer is not a good choice (see the picture on the previous page). Incidentally, the display on the previous page was made with an HP 16500A which has both logic analyzer and digitizing scope channels.

Logic analyzers are particularly useful when looking at time relationships or data on a bus – e.g. a microprocessor address, data, or control bus. It can decode the information on microprocessor buses and present it in a meaningful form (more on this when we talk about state analyzers). Generally, when you are past the parametric stage of design, and are interested in timing relationships among many signals and need to trigger on patterns of logic highs and lows, the logic analyzer is a good choice.



2 What's a Logic Analyzer?

Now that we've talked a little about when to use a logic analyzer, let's look in a bit more detail at what a logic analyzer is. Up to now, we've used the term "logic analyzer" rather loosely. In fact, most logic analyzers are really two analyzers in one. The first part is a timing analyzer, while the second part is a state analyzer. Each has specific functions that we will talk about in the following sections.

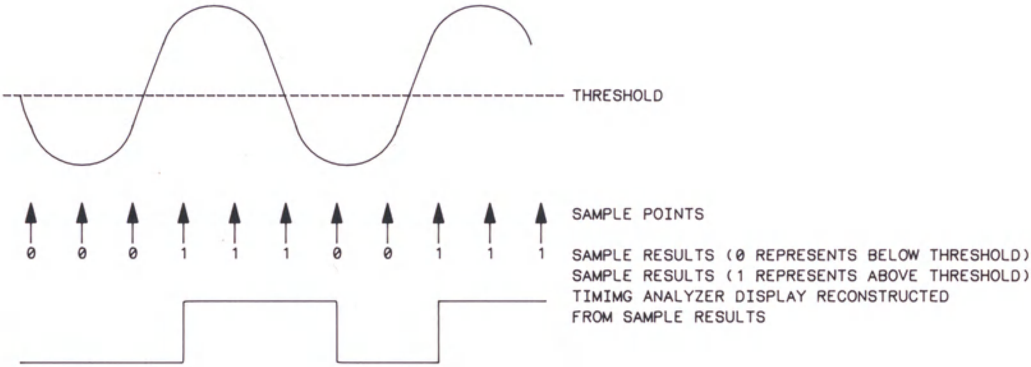


What's a Timing Analyzer?

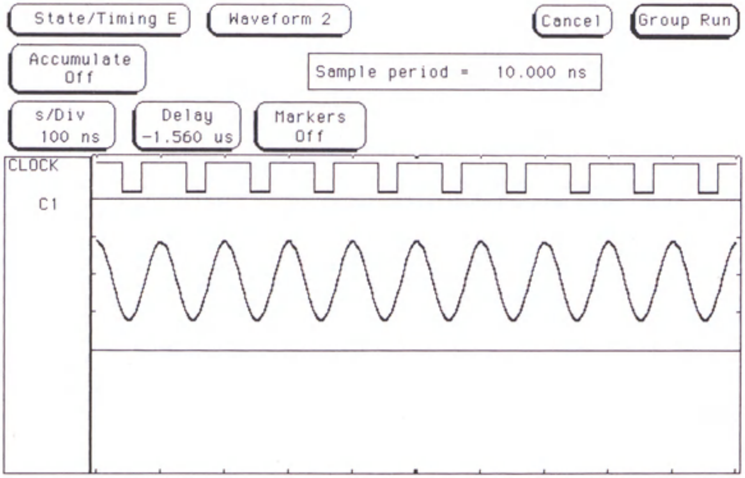
A timing analyzer is the part of a logic analyzer that is analogous to an oscilloscope. As a matter of fact, they can be thought of as close cousins. The timing analyzer displays information in the same general form as a scope, with the horizontal axis representing time and the vertical axis as voltage amplitude. Because the waveforms on both instruments are time-dependent, the displays are said to be in the "time domain."

Choosing the Right Sampling Method

A timing analyzer works by sampling the input waveforms to determine whether they are high or low. It cares about only one voltage-threshold. If the signal is above threshold when it is sampled, it will be displayed as a 1 or high by the analyzer. By the same criterion, any signal sampled that is below threshold is displayed as a 0 or low. From these sample points, a list of ones and zeros is generated that represents a one-bit picture of the input waveform. As far as the analyzer is concerned, the waveform is either high or low— no intermediate steps. This list is stored in memory and is also used to reconstruct a one-bit picture of the input waveform, as shown below. A timing analyzer is like a digitizing scope with only one bit of vertical resolution. With one bit of resolution, you can display only two states— high or low.

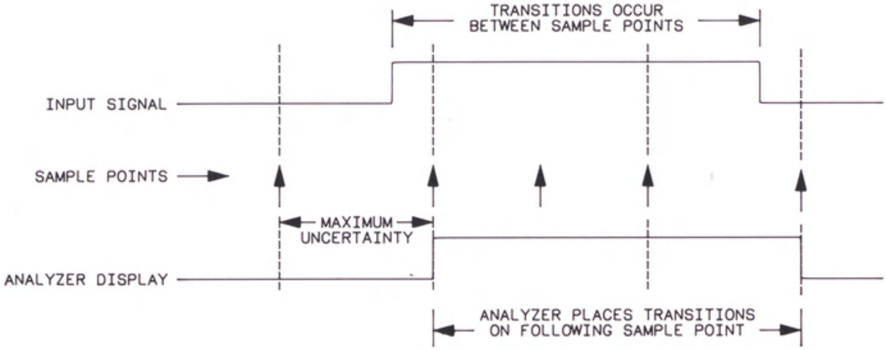


Notice the display shown below. These displays are actually the same signal (a sine wave) displayed by a digitizing scope and a timing analyzer.



This tendency to square everything up would seem to limit the usefulness of a timing analyzer. We should remember, however, that it is not intended as a parametric instrument. If you want to check rise time of a signal with an analyzer, you are using the wrong instrument. But if you need to verify timing relationships among several lines by seeing them all together, a timing analyzer is the logical (no pun intended) choice. For example, imagine that we have dynamic RAM in a system that must be refreshed every 2 ms. To ensure that everything in memory is refreshed within that 2 ms, a counter is used to count up sequentially through all rows of the RAMs and refresh each. If we want to make certain that the counter does indeed count up through all rows before starting over, a timing analyzer can be set to trigger when the counter starts and display all of the counts. Parametrics are not of great concern here— we merely want to check that the counter counts from 1 to N and then starts over.

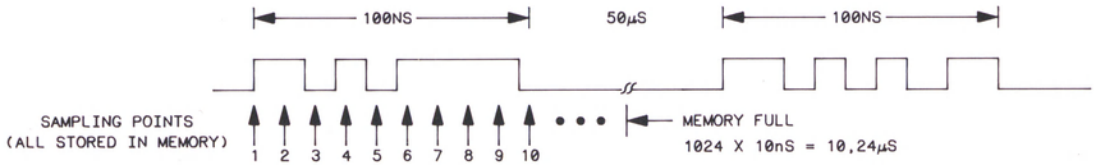
When the timing analyzer samples an input line, it is either high or low. If the line is at one state (high or low) on one sample and the opposite state on the next sample, the analyzer "knows" that the input signal transitioned at sometime in between the two samples. It doesn't know when, so it places the transition point at the next sample, as shown below. This presents some ambiguity as to when the transition actually occurred and when it is displayed by the analyzer. Worst case for this ambiguity is one sample period assuming that the transition occurred immediately after the previous sample point.



With this technique however, there is a trade-off between resolution and total acquisition time. Remember, that every sampling point uses one memory location. Thus the higher the resolution (faster sampling rate), the shorter the acquisition window.

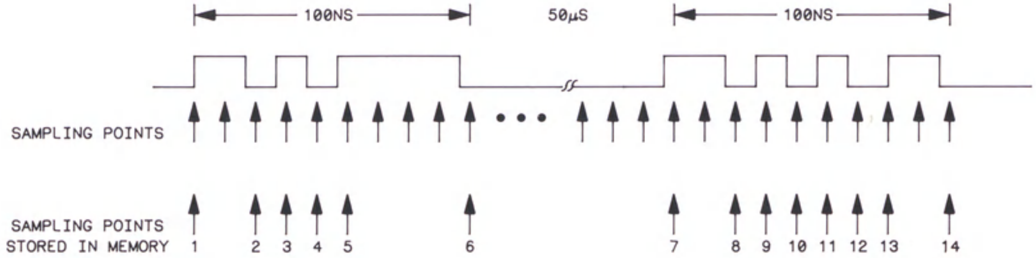
Transitional Sampling

When we capture data on an input line with data bursts like in the example below, we have to adjust the sampling rate to high resolution (e.g. 10 ns) to capture the fast pulses at the beginning. This means however, that a timing analyzer with 1 k (1024 samples) memory would stop acquiring data after 10.24 μs and the second data burst could not be captured.

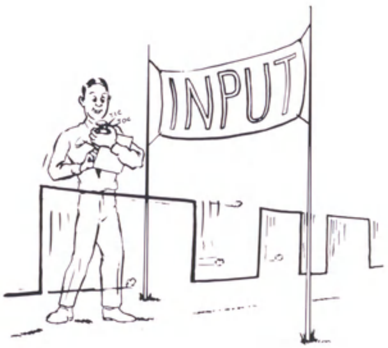


Note that we sample and store data for a long time where there is no activity. This uses up logic analyzer memory without providing additional information. All we really need to know is when these transitions occur and if they are positive or negative. Transitional timing, however, uses memory efficiently.

To accomplish this, we could use a "transition detector" at the input of the timing analyzer together with a counter. The timing analyzer will now store only those samples which were preceded by a transition, together with the elapsed time from the last transition. Thus we use only two memory locations per transition and no memory at all if there is no activity at the input. This technique is called "transitional timing" and is used in Hewlett-Packard's 1650/16500 family of logic analyzers. In our example we can not only capture the second burst, but also the third, fourth and fifth, depending how many pulses per burst are present. At the same time, we can keep the timing resolution as high as 10 ns.

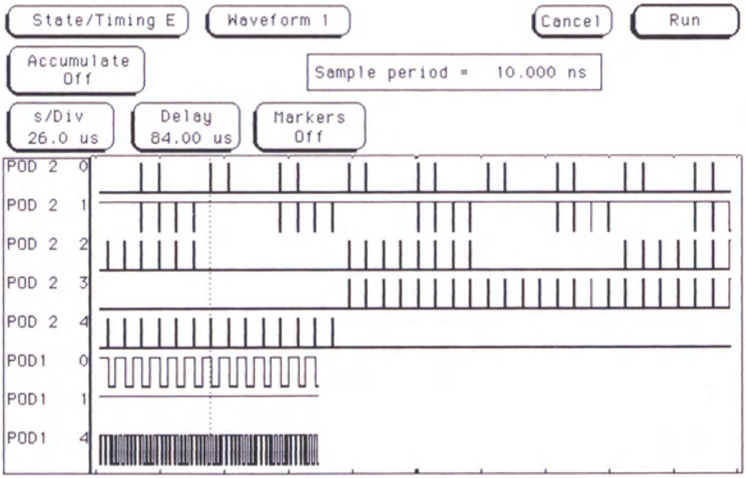


ONLY 28 MEMORY LOCATIONS REQUIRED (14 SAMPLING POINTS + 14 TIME INTERVALS)



We can now talk about "effective memory depth" which equals the total time data captured divided by the sampling period (10 ns). On the example at the right, which shows typical microprocessor activity, the total time data captured is 260 μs (26 μs/div x 10 div) at a sampling period of 10 ns which equals to an "effective memory depth" of 26 k.

Note:
This is a conceptual description of the transitional timing technique.



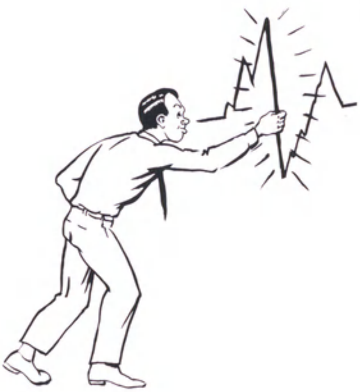
Glitch Capture

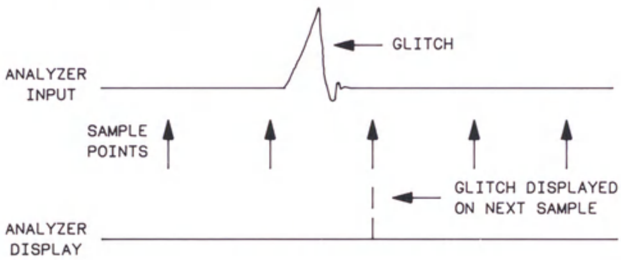
One headache of digital systems is the infamous "glitch." Glitches have a nasty habit of showing up at the most inopportune times with the most disastrous result. How do you capture a glitch that occurs once every 36 hours and sends your system into the weeds? Once again the timing analyzer comes to the rescue! Hewlett-Packard analyzers have glitch capture and trigger capability that makes it easy to track down elusive glitch problems.

A glitch can be caused by capacitive coupling between traces, power supply ripples, high instantaneous current demands by several devices, or any number of other events. Because they are difficult for most oscilloscopes to differentiate from valid transitions, a scope is generally not helpful in tracking down a glitch. However, since a timing analyzer samples the incoming data and can keep track of any transitions that occur between samples, it can readily recognize a glitch. In the case of an analyzer, a glitch is defined as any transition that crosses logic threshold more than once between samples.

The analyzer already keeps track of all single transitions that occur between samples, as we discussed before. To recognize a glitch, we "teach" the analyzer to keep track of all multiple transitions and display them as glitches.

While displaying glitches is a useful capability, it can also be helpful to have the ability to trigger on a glitch and display data that occurred before it. This can help us to determine what caused it. This capability also enables the analyzer to capture data only when we want it – when the glitch occurred. Think about the example we mentioned in the beginning paragraph of this section. We have a system that crashes periodically because a glitch appears on one of the lines. Since it occurs infrequently, to store data all the time (assuming we had enough storage capability) would result in an incredible amount of information to sort through. Another alternative is to use an analyzer without glitch trigger capability and sit in front of the machine pressing the RUN button and waiting until you see the glitch. Unfortunately, neither of the above are practical alternatives. If we can tell the analyzer to trigger on a glitch, it can stop when it finds one, capturing all the data that happened before it. We let the analyzer be the babysitter and when the system crashes, we have a record of what led up to the error.



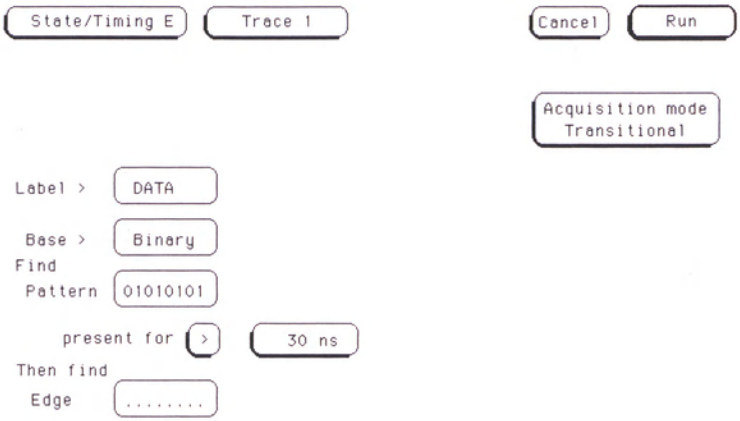


Triggering the Timing Analyzer

Another term that should be familiar to oscilloscope users is "triggering." It is also used in logic analyzers, but is often called "trace point." Unlike an analog oscilloscope which starts the trace right after the trigger, a logic analyzer continuously captures data and stops the acquisition after the trace point is found to display the data. Thus a logic analyzer can show information prior to the trace point, which is known as negative time, as well as information after the trace point.

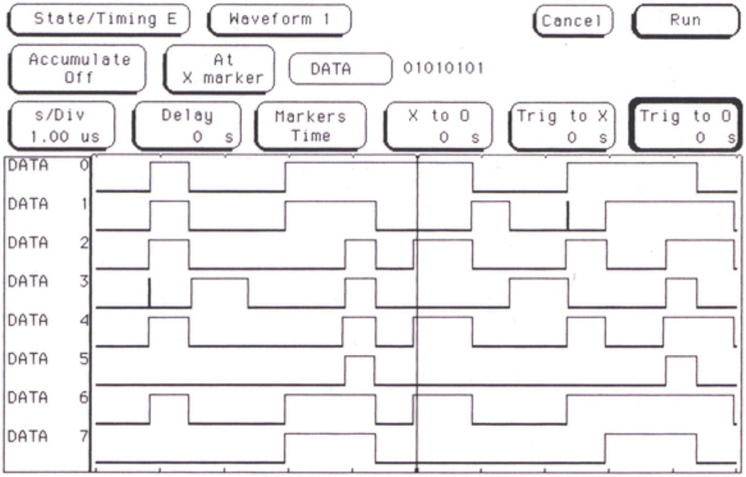
Pattern Trigger

Setting trace specifications on a timing analyzer is a bit different than setting trigger level and slope on an analog oscilloscope. Many analyzers trigger on a pattern of highs and lows across input lines. Notice the trace menu.



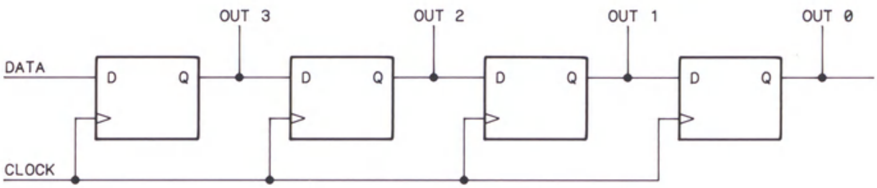
We have told the analyzer to start capturing data when channels 0, 2, 4, and 6 are high (logical 1) and when channels 1, 3, 5, and 7 are low (logical 0). The picture below shows the resulting display with the line in the middle indicating the trace point. At the trace point channels 0, 2, 4, and 6 are all high while channels 1, 3, 5, and 7 are low.

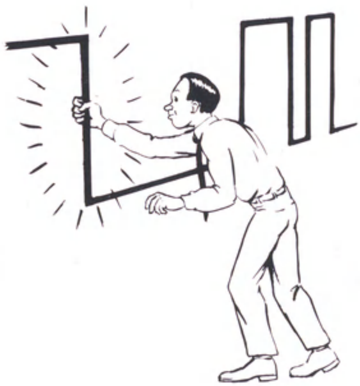
To make things easier for some users, the trigger point on most analyzers can be set not only in binary (1's and 0's) but in hex, octal, ASCII, or decimal. For instance, to set the previous example in hex the trigger specification would be 55 instead of 0101 0101. Using hex for the trigger point is particularly helpful when looking at buses that are 4, 8, 16, 24, or 32 bits wide, Imagine how cumbersome it would be to set a specification for a 24-bit bus in binary!



Edge Trigger

Edge triggering is a familiar concept to those accustomed to using an analog oscilloscope. When adjusting the "trigger level" knob on a scope, you could think of it as setting the level of a voltage comparator that tells the scope to trigger when the input voltage crosses that level. A timing analyzer works essentially the same on edge triggering except that the trigger level is preset to logic threshold. Why include edge triggering in a timing analyzer? While many logic devices are level-dependent, clock and control signals of these devices are often edge-sensitive. Edge triggering allows you to start capturing data as the device is clocked. As a simple example, take the case of an edge-triggered shift register that is not shifting data correctly. Is the problem with the data or the clock edge? In order to check the device, we need to verify the data when it is clocked— on the clock edge. The analyzer can be told to capture data when the clock edge occurs (rising or falling) and catch all of the outputs of the shift register. Of course, in this case we would have to delay the trace point to take care of the propagation delay through the shift register.





What's a State Analyzer?

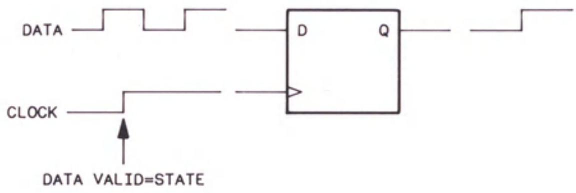
If you've never used a state analyzer, you may think it's an incredibly complex instrument that would take a large time investment to master. "Besides", you say to yourself, "What use could I have for a state analyzer? I design hardware." The truth is, many hardware designers find a state analyzer to be a very valuable "tool" in their "toolbox", especially when tracking down little bugs in software or hardware. It can eliminate "finger-pointing" between hardware and software teams when a problem does come up. And the state analyzer is not any more difficult to understand than a timing analyzer.

In the first part of this book we talked about one of two major parts of a logic analyzer— the timing analyzer. It is like a digitizing oscilloscope in some respects, especially when displaying timing waveforms. Since the horizontal axis on the waveform display is time, we say that it is in the "time domain."

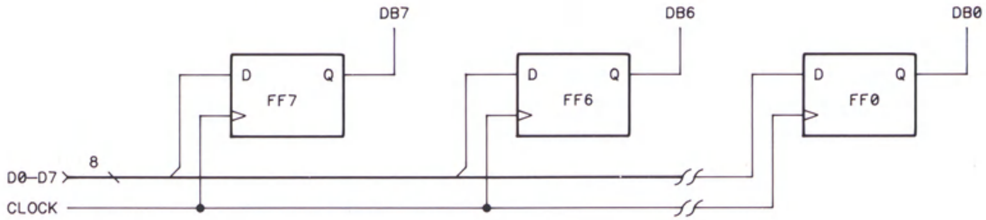
When Should I Use a State Analyzer?

If we want to understand when to use a State Analyzer, we need to know first what is a "state". A "state" for a logic circuit is a sample of a bus or line when its data is valid.

For example, take a simple "D" flip-flop, like the one shown below. Data at the "D" input will not be valid until a positive-going clock edge comes along. Thus, a state for the flip-flop is when the positive clock edge occurs.



Now imagine that we have eight of these flip-flops in parallel. All eight are connected to the same clock signal. When a positive transition occurs on the clock line, all eight will capture data at their "D" inputs. Again, a state occurs each time there is a positive transition on the clock line. These eight lines are analogous to a microprocessor bus.



If we connected a state analyzer to these eight lines and told it that a positive transition on the clock line is when we want to collect data, the analyzer would do just that. Any activity on the inputs will not be captured by the state analyzer unless the clock is going high.

This points up the major difference between a timing and a state analyzer. The timing analyzer has an internal clock to control sampling, so it asynchronously samples the system under test. A state analyzer sychronously samples the system since it gets its sampling clock from the system.

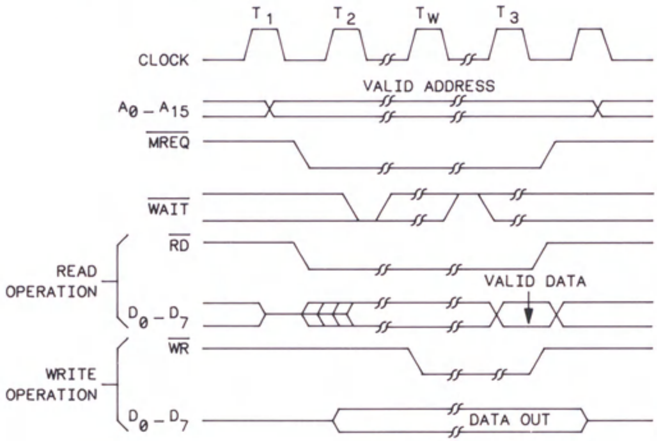
As a rule of thumb, you might remember to use a state analyzer to check "what" happened on a bus and a timing analyzer to see "when" it happened. Therefore, a state analyzer generally displays data in a listing format and a timing analyzer displays data as a waveform diagram. We have to be extremely careful not to misinterpret the data when the logic analyzer is capable of displaying state data as a waveform diagram and timing data as a listing.

Understanding Clocks

In the timing analyzer, sampling is under direction of a single internal clock. That makes things very simple. However, in the world of microprocessors, a system may have several "clocks." Let's look at a brief example.

Suppose for a moment that we want to trigger on a specific address in RAM and see what data is stored there. Further, we'll assume that the system uses a Zilog Z80. During a read or write cycle, the Z80 first puts an address on the address bus. Next it asserts MREQ, showing that the address is valid for a memory read or write. Last, the RD or WR line is asserted, depending on whether we are doing a read or write. The WR line is asserted only after the data on the bus is valid.

In order to capture addresses from the Z80 with our state analyzer, we will want to capture when MREQ line goes low. But to capture data, we will want the analyzer to sample when the WR line goes low (write cycle) or when RD goes high (read cycle).



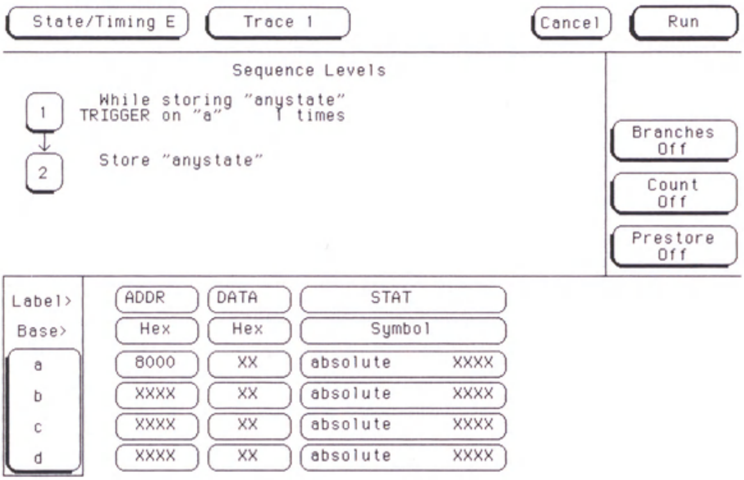
Some microprocessors multiplex data and address on the same lines. The analyzer must be able to clock in information from the same lines but with different clocks. This, in essence, acts as a demultiplexer to capture an address at the proper time and then catch data that occurs on the same lines.

Triggering the State Analyzer

A state analyzer still gives the capability to qualify the data we want to store. If we are looking for a specific pattern of highs and lows on the address bus, we can tell the analyzer to start storing when it finds that pattern and to continue storing until the analyzer's memory is full.

In the following example, we have set the trigger point as 8000H (hexadecimal). In this case we want to find out what is in location 8000H, so we set the data trigger as don't cares (xx). This tells the analyzer to trigger on address 8000H regardless of what the data is at that point.

The analyzer captured address 8000H and all following states. Notice that data is C3H at address 8000. Notice that all of the information is displayed in hexadecimal format. We could display it in binary, if that is helpful. However, it may be more helpful to have the hex decoded into assembly code.



Disassembly

```
State/Timing E    Listing 1                Cancel    Run

Markers
  Off

Label>  ADDR    DATA    STAT
Base>   Hex     Hex     Symbol

  0     8000    C3      OPCODE FETCH
  1     8001    50      MEMORY READ
  2     8002    80      MEMORY READ
```

If you specify that all information on the buses is to be displayed in hex, you will get a display that resembles the one shown below. What do these hex codes mean? In the case of a processor, specific hex characters comprise an instruction. If you are very familiar with the hex codes, you may be able to look at a hex listing like the above and know what instruction is represented by it. Most of us, however, can't do that. For that reason, most analyzer makers have designed software packages called disassemblers or inverse assemblers. The job of these packages is to translate the hex codes into assembly code to make them easier to read. For example, the display shown above has C3, 50, 80, and C3 shown. If we look those codes up in the 8085 manual we find that they represent JMP 50 80 (jump to location 8050), and then another JMP instruction (C3). Rather than having to look each code up, the inverse assembler does it for us. Look at the following display and notice the difference.

```
State/Timing E    Listing 1                Cancel    Run

Markers
  Off

Label>  ADDR    8085 Mnemonic         STAT
Base>   Hex     hex                   Symbol

  0     8000    JMP   8050            OPCODE FETCH
  1     8001      50 memory read      MEMORY READ
  2     8002      80 memory read      MEMORY READ
```

Understanding Sequence Levels

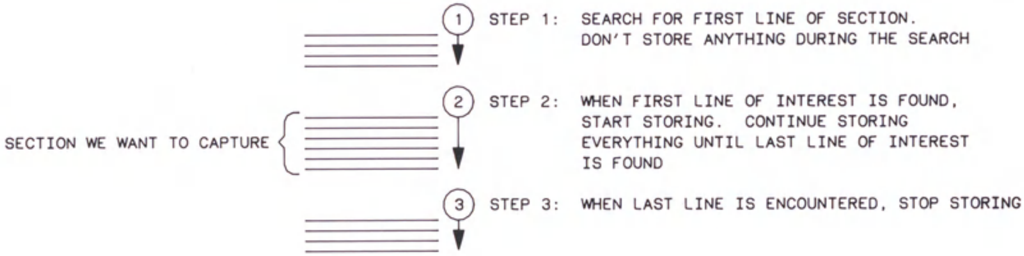
State analyzers have "sequence levels" that aid triggering and storage. Sequence levels allow you to qualify data storage more accurately than a single trigger point. This means that you can more accurately window in on the data without storing information you don't need. Sequence levels usually look something like this:

1 find xxxx
   else on xxxx go to level x
2 then find xxxx
   else on xxxx go to level x
3 trigger on xxxx

Sequence levels are especially useful for getting into a subroutine from a specific point in the program.

Selective Storage Saves Memory and Time!

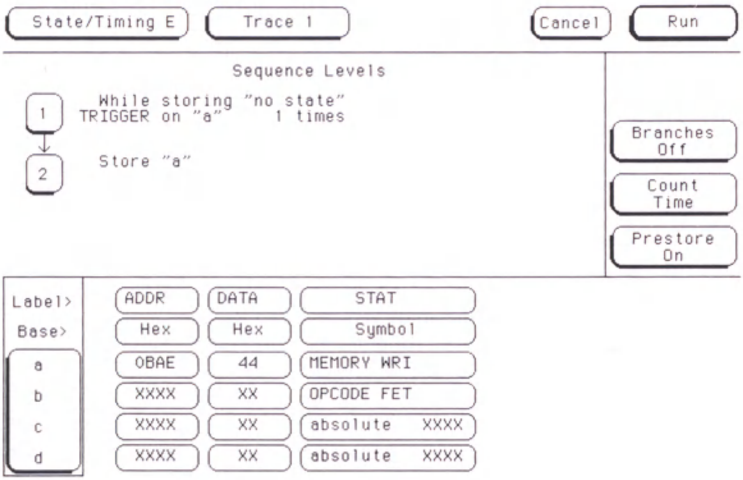
Sequence levels make possible what we call selective storage. Selective storage simply means storing only a portion out of a larger whole. For instance, suppose we have an assembly routine that calculates the square of a given number. If the routine is not calculating the square correctly, we can tell the state analyzer to capture that routine. We do this by first telling the analyzer to find the start of the routine. When it does find the start address, we then tell it to look for the ending address while storing everything in-between. When the end of the routine is found, we tell the analyzer to stop storing (store no states). The following diagram shows how selective storage works.



Application Example – Prestore

Beyond these features, the 1650/16500 family of Hewlett-Packard's logic analyzers lets you further specify a so-called "prestore" condition. This means that the analyzer will store the last two states in addition to the one stored anyway. But let's have a look at how this can help you in solving some of your problems more easily. The nightmare of every hardware and software designer is when a global memory location of his system gets overwritten with erratic data from time to time. In our example we assume that address 0BAEH is reserved for a global variable and gets intermittently overwritten with data 44H.

We can trigger the state analyzer on a memory write to address 0BAEH with data 44H ("a") and store only these conditions. Now we can see how often wrong data gets written to our global variable. To find the cause of the problem however, we'd need to know which instruction (opcode fetch) ("b") generated the erratic write cycle. If we specify 'prestore on "b"' where "b" is defined as an opcode fetch, we also store the last two instruction cycles prior to the write cycle, pointing to the problem. Thus we need only three state analyzer memory locations to capture the cause of the problem.



State/Timing E | Listing 1 | Cancel | Run

Markers Off

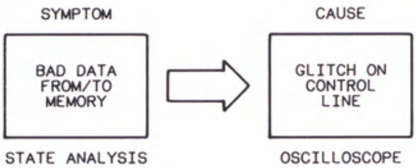
Label> / Base>	ADDR / Hex	8085 Mnemonic / hex	STAT / Symbol	Time / Relative
1	042D	JMP ****	OPCODE FET	PRESTORE
2	0431	PUSH PSW	OPCODE FET	PRESTORE
3	0BAE	44 memory write	MEMORY WRI	1.089 ms
4	042D	JMP ****	OPCODE FET	PRESTORE
5	0431	PUSH PSW	OPCODE FET	PRESTORE
6	0BAE	44 memory write	MEMORY WRI	1.089 ms
7	042D	JMP ****	OPCODE FET	PRESTORE
8	0431	PUSH PSW	OPCODE FET	PRESTORE
9	0BAE	44 memory write	MEMORY WRI	1.089 ms
10	042D	JMP ****	OPCODE FET	PRESTORE
11	0431	PUSH PSW	OPCODE FET	PRESTORE
12	0BAE	44 memory write	MEMORY WRI	1.089 ms
13	042D	JMP ****	OPCODE FET	PRESTORE
14	0431	PUSH PSW	OPCODE FET	PRESTORE
15	0BAE	44 memory write	MEMORY WRI	1.089 ms
16	0432	XRA A	OPCODE FET	PRESTORE

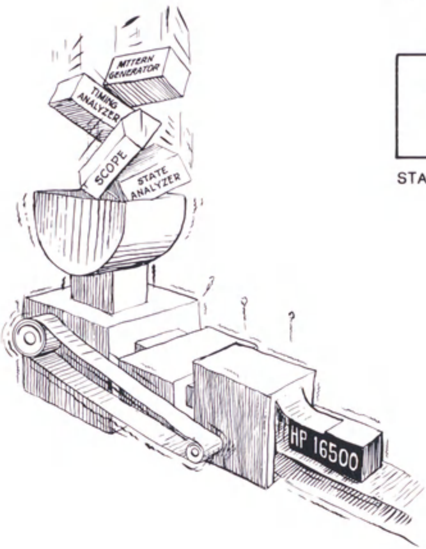
3 Put Them All In One Instrument!

So far we have talked about oscilloscopes, state and timing analyzers and their applications. If you are designing or servicing digital hardware, you probably have applications for each one of the tools in your area. In this section we'll talk about how to use these tools together to isolate the faults in your system faster and more efficiently.

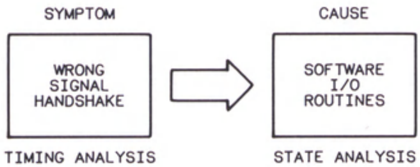
Symptoms and Their Cause

If you troubleshoot digital circuitry you often have to ask yourself: "What causes this symptom?" It might be quite easy to identify the symptom of a fault, but you need to find the cause to fix the problem. Many times, causes and symptoms are in different domains. For example, a glitch on a control line of a memory can cause wrong data to be at read from/written to memory. The symptom (wrong data) can be found in the data domain by using a state analyzer and triggering on the suspect memory address. The cause however, can not be identified in the data domain. A glitch is only identifiable in the time domain by using either a timing analyzer or a scope. In our example we are more interested in the size of the glitch (parameter) and therefore, the scope is the best tool for the measurement.



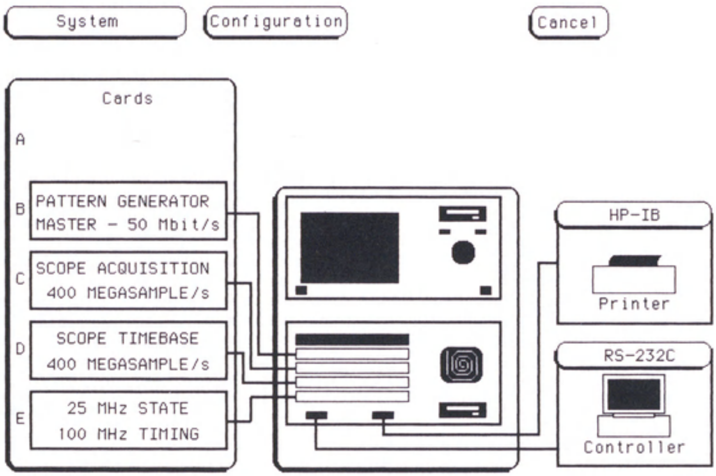


It is also possible that the symptom is in the time domain (e.g. a bad handshake signal on I/O lines), and the cause is in the data domain (wrong software I/O routine).



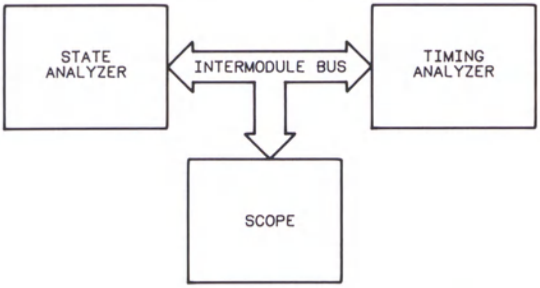
Intermodule Measurements

A measurement which involves more than one measurement instrument is called an "intermodule measurement". An intermodule measurement requires that all measurement tools are integrated in a single instrument and are able to capture data simultaneously. The figure below shows the system configuration menu from a HP 16500A Logic Analyzer System including state and timing analyzers as well as a digitizing oscilloscope and a pattern generator for stimulus.



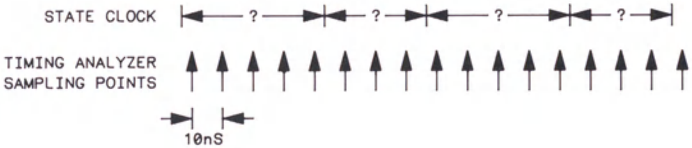
Cross-Module Triggering

In our examples we talked about triggering a module (state, timing analyzer or scope) on the symptom of the problem. Once the symptom occurs and the appropriate analyzer triggers, the module which monitors the cause has to start capturing data. This is achieved by arming one module from the trigger of the other. For full functionality it is necessary that each module can receive and send trigger signals. The bus, on which these trigger signals are transmitted, is called the "intermodule bus" or IMB.



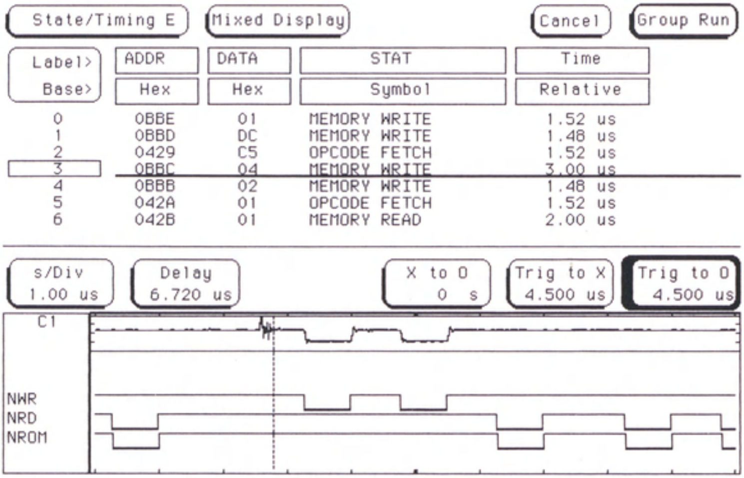
Cross-Module Time Correlation

Once we have successfully triggered all our measurement modules and finished data capture, we need to look at the captured data. We are all familiar with the waveform display of a scope and discussed how to present the data captured by a state or timing analyzer earlier. In order to correlate from one domain to another, it is convenient to display data from both domains on one screen. But how can we correlate between state and timing other than the trace point? Remember, the timing analyzer uses an internal sampling clock that is asynchronous to the system while the state analyzer samples synchronously to the target system. If we count the time between the external state samples, we have enough time information to correlate from any point of the timing analyzer waveform to the appropriate location of the state analyzer listing.



Application Example

In the figure below you see the measurement result from the example we used before. The state analyzer is used to trigger on a certain memory access. Both the timing analyzer and scope are triggered by the state analyzer to provide timing information over multiple channels as well as parametric information on fewer channels. Note that the cursors are used to correlate between time domain (scope and timing analyzer) and data domain (state analyzer).



4 How Do I Connect To My Target System?

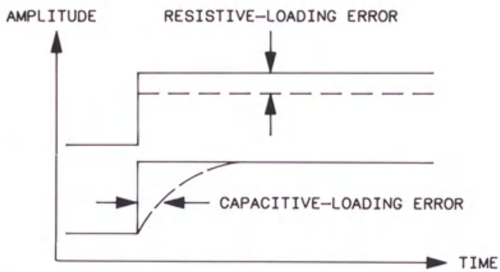
So far we've talked about some of the differences between scopes, and timing and state analyzers. Before we're ready to apply these new tools we should talk about one more subject, the probing system.

From using an oscilloscope, you're probably familiar with passive probes. A scope probe is designed to gain easy access to the target system while minimizing the signal distortion. Since we want to look at parametric information like voltage levels and risetimes, it is important that the probe doesn't load the circuit under test significantly. A typical scope probe has 1 MΩ impedance shunted by 10pF, depending on the bandwidth required.

On the other hand, a logic analyzer probe is designed to allow connection of a high number of channels to the target system easily by trading off amplitude accuracy of the signal under test. Remember, that a logic analyzer only distinguishes between two voltage levels! Traditionally, logic analyzers used active probe pods, which had the signal detection circuitry for eight channels integrated. From these pods, we could connect with leads to the circuit under test. The typical impedance of a logic analyzer probe is in the area of 100 kΩ shunted by 8 pF at the input of the active pod. The connecting wires, however, add another 8 pF stray capacitance,giving a total of 16 pF per channel.

Resistive vs Capacitive Loading

How does the probe impedance affect my measurement? Two sources of loading, resistive and capacitive are the main cause of signal distortion. Resistive loading affects the amplitude of the output through a resistive divider effect. Capacitive loading affects the timing of the signal under test by rounding and slewing the edges.



Amplitude errors from resistive loading are not significant enough to affect most circuit's performance, even when probing with HP's 54001A 1 GHz scope probes with 10 kΩ resistance. In fact, most logic families can operate correctly with as much as a 10% error in amplitude. Because most of these digital ICs exhibit typical output impedance in the low hundreds of ohms or less, you can use a probe with tip resistance measuring a few kΩ.

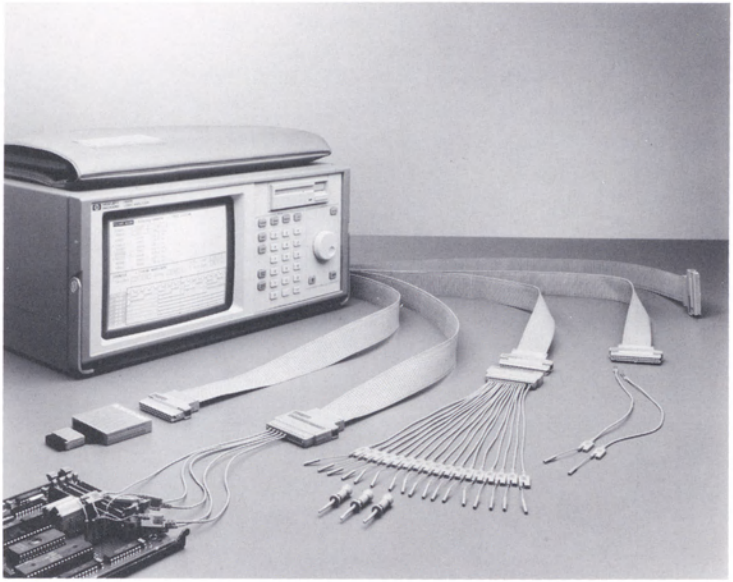
The capacitive loading of probes becomes more important as clock rates continue to increase in new designs. Because of this increase of clock rates, circuits are more sensitive to timing errors of even a few nanoseconds. The basic timing-error immunity, on the other hand, is limited by a circuit's clock rate. A CMOS circuit that drives a given load may operate correctly even with a higher clock rate, but the extra capacitive loading of a probe on that circuit can produce unexpected timing problems.

The following table shows typical increases in CMOS gate delay due to probe capacitance:

Capacitance	Standard CMOS Delta T	High Speed CMOS Delta T
15 pF	25 ns	2.5 ns
8 pF	13 ns	1.3 ns
2 pF	3 ns	0.3 ns

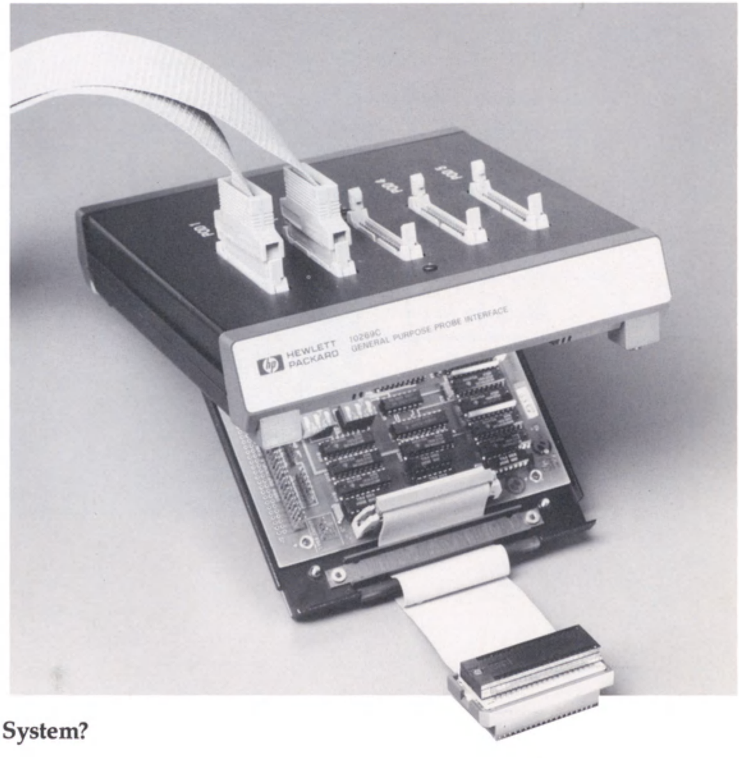
Passive Probing

With the 1650/16500 family of logic analyzers, Hewlett-Packard introduced a new probing concept. Instead of using active pods for groups of eight channels, HP designed a new passive probe with sixteen channels per cable. Each channel is terminated at both ends with 100 kΩ and 8 pF. You can best compare the new passive probe electrically with the good old scope probe. The advantage of the passive probing system, beside much smaller size and higher reliability, is that we can terminate the probe right at the point of connection to the target system. This avoids additional stray capacitance due to the wires from the larger active pods to the circuit under test. As a result, your circuit under test only "sees" 8 pF load capacitance instead of 16 pF with previous probing systems.



Preprocessors and Other Accessories

Connecting a state analyzer to a microprocessor system requires some effort in terms of mechanical connection and clock selection. Remember, we have to clock to the state analyzer whenever data or addresses on the bus are valid. With some microprocessors it might be necessary to use external circuitry to decode several signals to derive the clock for the state analyzer. A preprocessor provides not only fast, reliable and correct mechanical connection to your target system, but also the necessary electrical adaptation like clocking and demultiplexing to capture your system's operation correctly. Some microprocessores prefetch information from memory that may never get executed. Preprocessors can also distinguish prefetched information from executed. Furthermore, a preprocessor typically comes together with a disassembler to decode the hexadecimal information into microprocessor mnemonics, as discussed earlier.



Summary

We hope this booklet has helped you gain a better understanding of what a logic analyzer is and does. Since most analyzers are made up of two major parts, timing and state, we have covered them separately. But together, they make up a powerful tool for the digital designer.

The timing analyzer is closely akin to the oscilloscope, but is better suited to bus-type structures or applications where you are dealing with many lines. It also has the ability to trigger on patterns among the lines, or even glitches.

A state analyzer is most often viewed as a software tool. In reality it also has many uses in the hardware domain. Since it gets its clock from the system under test, it can be used to catch data when the system sees it— on the system's clock.

Armed with this fundamental knowledge, we believe you can use a logic analyzer with confidence.

For more information about logic analyzers and what they can do for you, call your nearest Hewlett-Packard Sales Office.

For more information, call your local HP sales office listed in the telephone directory white pages. Ask for the Electronic Instruments Department. Or write to **Hewlett-Packard, U.S.A.:** P.O. Box 10301, Palo Alto, CA 94303-0890. **Europe/Middle East/Africa:** Central Mailing Department, P.O. Box 1180 AM Amstelveen, The Netherlands. **Canada:** 6877 Goreway Drive, Mississauga, L4V1M8, Ontario. **Japan:** Yokogawa-Hewlett-Packard, Ltd., 3-29-21, Takaido-Higashi, Suginami-ku, Tokyo 168. Elsewhere in the world, write to: Hewlett-Packard Intercontinental, 3495 Deer Creek Road, Palo Alto, CA 94304.

HEWLETT PACKARD

5954-2686

April 1988

Printed in U.S.A.